Technical Report

A 12*(1+|R|/(4m))-speed algorithm for scheduling constrained-deadline sporadic real-time tasks on a multiprocessor comprising m processors where a task may request one of |R| sequentially-reusable shared resources

Björn Andersson

Arvind Easwaran

HURRAY-TR-100201

Version:

Date: 02-03-2010

A 12*(1+|R|/(4m))-speed algorithm for scheduling constrained-deadline sporadic real-time tasks on a multiprocessor comprising m processors where a task may request one of |R| sequentially-reusable shared resources

Björn Andersson, Arvind Easwaran

IPP-HURRAY!

Polytechnic Institute of Porto (ISEP-IPP)

Rua Dr. António Bernardino de Almeida, 431

4200-072 Porto

Portugal

Tel.: +351.22.8340509, Fax: +351.22.8340509

E-mail:

http://www.hurray.isep.ipp.pt

Abstract

We present a 12*(1+|R|/(4m))-speed algorithm for scheduling constrained-deadline sporadic real-time tasks on a multiprocessor comprising m processors where a task may request one of |R| sequentially-reusable shared resources.

A 12*(1+|*R*|/(4*m*))-speed algorithm for scheduling constrained-deadline sporadic real-time tasks on a multiprocessor comprising *m* processors where a task may request one of |*R*| sequentially-reusable shared resources

Björn Andersson and Arvind Easwaran
CISTER-ISEP Research Centre
Polytechnic Institute of Porto, Portugal
{bandersson,aen}@dei.isep.ipp.pt

Abstract

We present a 12(1+|R|/(4m))-speed algorithm for scheduling constrained-deadline sporadic real-time tasks on a multiprocessor comprising m processors where a task may request one of |R| sequentially-reusable shared resources.*

1. Introduction

Consider the problem of preemptively scheduling a task set τ of n constrained-deadline sporadic tasks (τ_1 to τ_n) on m identical processors (P_1 to P_m). A task generates a (potentially infinite) sequence of jobs, occurring at least T_i time units apart. A job by τ_i requires up to C_i units of execution over the next D_i time units after its arrival (with T_i, D_i, C_i) being real numbers and $0 \leq C_i \leq D_i \leq T_i$. A processor executes at most one job at a time and no job may execute on multiple processors simultaneously.

There is a set R of $|R|$ resources. These resources are not processors; instead they are shared resources (for example shared I/O devices or shared data structures) which a task needs in addition to a processor for execution of part of its execution, typically for performing an operation on the resource. We assume that a task may request a resource; the resource request may be granted at that time or later, and then the task may release the resource. We say that a task which has been granted a resource *holds* the resource until the task has released the resource. We assume that a job of task τ_i may perform at most one request for a resource in R. Let r^k denote a resource; clearly we have $r^k \in R$. We let C_i^k denote the amount of units of execution that a job of task τ_i needs to execute with resource r^k from when the time that the request was granted until the task released the resource.

We assume that resources must be held under mutual exclusion, that is, it is not allowed that two different jobs hold the same resource simultaneously. We make no assumption where in the execution of a job that the request for a resource is made; for example we allow for the possibility that one job of task τ_i requests a resource r^k in the beginning of its execution but another job of task τ_i requests a resource r^k in the end of its execution. Since a job can make only a single request for a resource, it clearly follows that nested resource requests cannot occur. Clearly, a task whose resource request is not granted cannot execute; it must wait and during those time units until the task gets granted the resource, the task does not perform any units of execution.

We will find it useful to speak about the speed of a processor. For a processor of speed s, it holds that if a task executes for L time units on this processor then the task completes $s*L$ units of execution. As an illustration, if a job of task τ_i executes for C_i/s time units on a processor of speed s then it completes C_i units of execution and hence the job finishes. Note that this concept of speed also applies to resources: if a job of task τ_i has just been granted resource r^k and this job executes for C_i^k/s time units on a processor of speed s then it completes C_i^k units of execution and hence the request of the job for resource r^k finishes.

The research literature has made available several solutions for this problem. For a single processor system, the protocols PCP [1] and SRP [2] together with RM [3] and EDF [3] provide very good performance in terms of being able to guarantee that deadlines are met before run-time (schedulability). They use a scheduling algorithm that originally was designed for tasks which do not share resources and these works augment such an algorithm with a resource-sharing protocol which changes the priority of a task when certain events occur that are related to resource sharing (for example, a resource request is not immediately granted). For a multiprocessor system, solutions have been developed for which it is possible to prove before run-time that deadlines will be met. But no such solution with provably good performance (in terms of having been proven to have a finite competitive ratio) is known. We say that an algorithm A has competitive ratio CPT_A if, for every task set τ for which it is possible to meet deadlines, it holds that if the speed of processors are multiplied by CPT_A then algorithm A will meet deadlines as well. We also say, as a shorthand expression, that an algorithm A with competitive ratio CPT_A is a CPT_A-speed algorithm.

We believe that the current non-existence of multiprocessor real-time scheduling algorithms with resource sharing with finite competitive ratio suggests that

the problem of multiprocessor real-time scheduling with resource sharing is currently not well understood and because of its practical significance (multicore processors are becoming commonplace now and resource sharing may potentially limit the use of its parallel processing units), it ought to be understood better.

Therefore, in this paper, we present a new algorithm (called *gEDF-vpr*) for scheduling real-time tasks which share resources. The new algorithm has competitive ratio $12*(1+|R|/(4m))$.

The main idea of the new algorithm is as follows. We use m processors to emulate $2m+|R|$ virtual processors. There are m virtual processors of type-1; there are $|R|$ processors of type-2 and there are m processors of type-3. The time window from the arrival of a job until its absolute deadline can be split into three equal-length sub-windows. The first sub-window is used for execution without a resource and before a resource request has been made; this execution is done on virtual processors of type-1. The second sub-window is used for execution when a resource is requested and this request has not yet been completed; this execution is done on virtual processors of type-2. The third sub-window is used for execution without a resource and after a resource request has been completed. The benefit of this approach is that the scheduling problem with resource sharing (and particularly its analysis) can be simplified by decomposing it into three scheduling problems where resource sharing does not exist (which is the case on type-1 and type-3) or where resource sharing exist but it is easier to deal with (which is the case on type-2 where each virtual processor serves its dedicated resource).

Since we want to prove that the new algorithm has a competitive ratio of $12*(1+|R|/(4m))$, it follows that for such a proof, we only need to consider task sets where for each task τ_i it holds that $C_i/\min(D_i, T_i) \leq 1/(12*(1+|R|/(4m)))*s$ and if it requests resource k then it also holds that for $C_i^k/\min(D_i, T_i) \leq 1/(12*(1+|R|/(4m)))*s$, where s is the speed of the processors used by *gEDF-vpr*.

We believe this result is interesting because last Dagstuhl seminar on scheduling (in 2008) stressed multiprocessor scheduling with shared resources as an open problem [4]. We also believe it may be interesting at this upcoming Dagstuhl scheduling seminar 2010 to discuss possible ways to (i) attain greater performance and (ii) eliminate the factor $|R|$ in the expression of the competitive ratio. We also hope it will stimulate discussions (between real-time/operations research/theoretical computer science communities) on the design of suitable frameworks for multiprocessor scheduling with resource sharing.

The remainder of this paper is organized as follows. Section 2 presents the new algorithm. Section 3 presents notation and results we will use. Section 4 presents the proof of the competitive ratio of the new algorithm.

2. The new algorithm

We call this algorithm *gEDF-vpr*. This algorithm is only defined for task sets where for each task τ_i it holds that $C_i/\min(D_i, T_i) \leq 1/(12*(1+|R|/(4m)))*s$ and if it requests resource k then it also holds that for $C_i/\min(D_i, T_i) \leq 1/(12*(1+|R|/(4m)))*s$, where s is the speed of the processors used by *gEDF-vpr*. If *gEDF-vpr* is used on task sets for which these conditions are not true, then the algorithm declares failure and this should be interpreted as that *gEDF-vpr* caused a deadline miss.

Figure 1 illustrates the main idea of *gEDF-vpr*. The main idea is to use m processors to emulate a larger number of virtual processors. These virtual processors are partitioned into different types and a certain type serves a certain phase of the execution of a job.

By using m processors of speed 1, we can emulate $2m+|R|$ *virtual processors*, specifically these ones:

1. m virtual processors of speed $2m/(4m+|R|)$ and
2. $|R|$ virtual processors of speed $m/(4m+|R|)$ and
3. m virtual processors of speed $2m/(4m+|R|)$

We say that a virtual processor is either of type-1, type-2 or type-3 respectively. We give them indices so that

virtual processors of type 1 are given identifiers $1..m$ and

virtual processors of type 2 are given identifiers $m+1..m+|R|$ and

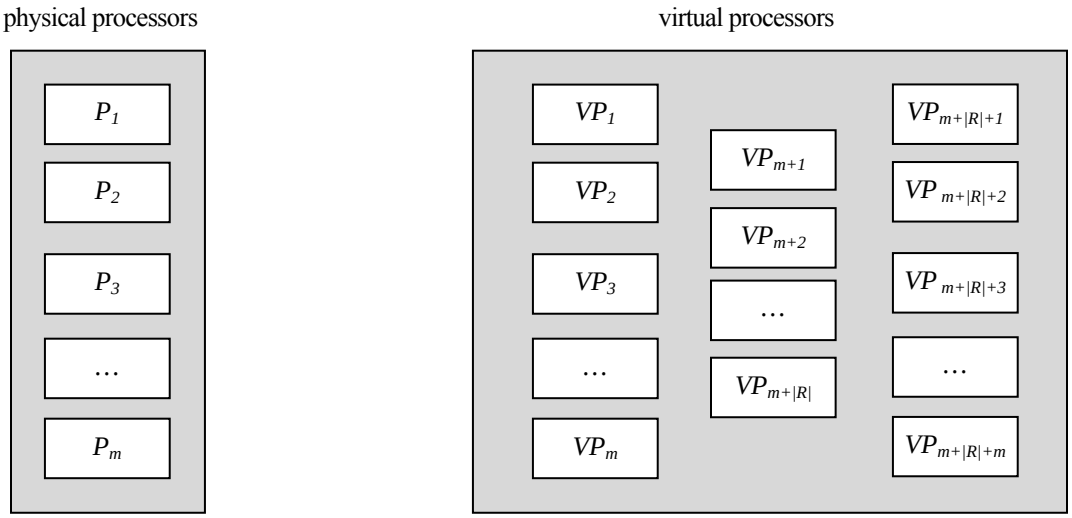
virtual processors of type 3 are given identifiers $m+|R|+1..2m+|R|$.

A task τ_i is at every instant assigned to exactly one phase. When a job arrives, the job is assigned phase-1. $D_i/3$ time units later, it is assigned phase-2. Additional $D_i/3$ time units later, it is assigned phase-3 and it is supposed to (in order to meet deadlines) to finish execution within $D_i/3$.

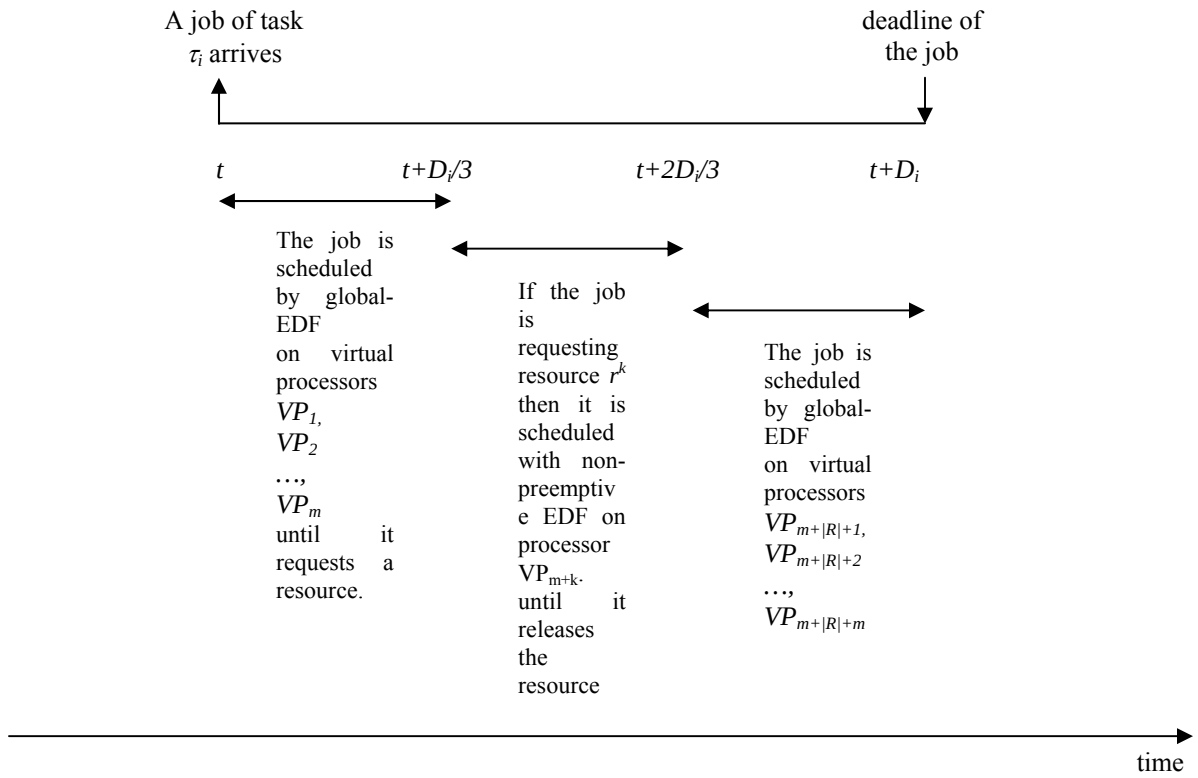
A task performing no resource request is only in phase-1 and executes only on virtual processors of type-1. For tasks which make resource requests, the following apply:

- A task τ_i which is in phase-1 is ready for execution only if it has not yet made a resource-access request.
- A task τ_i which is in phase-2 is ready for execution only if its resource access has not yet been granted and completed.
- A task τ_i which is in phase-3 is ready for execution only if it has remaining unfinished execution to perform.

We schedule (i) all phase-1 tasks onto virtual processors of type-1 using gEDF with the absolute deadline of task τ_i being computed based on $D_i/3$, (ii) a phase-2 task which request resource r^k is assigned virtual processor $m+k$ and non-preemptive EDF scheduling is used there with the absolute deadline of task τ_i being computed based on $D_i/3$ and (iii) all phase-3 tasks onto processors of type-3 using gEDF with the absolute deadline of task τ_i being computed based on $D_i/3$.



(a) Virtual processors of lower speed are formed out of physical processors.



b) **Different phases of a execution of a job are dispatched on different virtual processors.**

Figure 1. An illustration of the new algorithm *gEDF-vpr*.

3. Notations and Results we will use

We let *sched*(A, τ, m, s) denote a predicate meaning that the task set τ meets all deadlines when scheduled by algorithm A on m processors of speed s. The term "meets deadlines" here should be interpreted as "meet deadlines for every possible arrival of tasks which is possible as given by the parameters of the task set τ".

We let $TD(\tau)$ denote a function which takes a task set τ as parameter and outputs a task set which differs from τ only in that for each task τ_i in $TD(\tau)$, the parameter D_i is one third of the parameter D_i of the its corresponding task in τ.

We let $TD1(\tau)$ denote a function which takes a task set τ as parameter and outputs a task set which differs from τ only in that for each task τ_i in $TD1(\tau)$, the parameter D_i is one third of the parameter D_i of the its corresponding task in τ and we also set $C_i^k=0$ for every resource.

We let $TD2(\tau)$ denote a function which takes a task set τ as parameter and outputs a task set which differs from τ only in that for each task τ_i in $TD2(\tau)$, the parameter D_i is one third of the parameter D_i of the its corresponding task in τ and we also set $C_i=C_i^k$, where k is the resource that task τ_i may request.

We let $TD3(\tau)$ denote a function which takes a task set τ as parameter and outputs the a task set which differs from τ only in that for each task τ_i in $TD3(\tau)$, the parameter D_i is one third of the parameter D_i of the its corresponding task in τ and we also set $C_i^k=0$ for every resource.

It can be noted that $TD1(\tau)$ is identical to $TD3(\tau)$ so one of these definitions is actually redundant. We choose to define both of them however because we will use both of them in different context (in Section 4) when we prove the competitive ratio of a new algorithm. We can also intuitively understand the meaning of "TD" as "one ThirD".

We know from Theorem 2.2 in [5] that if it is possible to meet deadlines then gEDF meets deadlines as well if provided processors that are twice as fast.

4. Proving the competitive ratio of the new algorithm

Clearly, we have:

$$(\exists A: \text{sched}(A, \tau, m, 1)) \Rightarrow (\exists A: \text{sched}(A, TD(\tau), m, 3)) \quad (1)$$

because processors that are three times as fast makes it possible to meet deadlines that are one third of the deadlines of the original task set.

We also have:

$$(\exists A: \text{sched}(A, TD(\tau), m, 3)) \Rightarrow (\exists A: \text{sched}(A, TD1(\tau), m, 3)) \quad (2)$$

because the right hand side reduces the demand on resources.

We also have:

$$(\exists A: \text{sched}(A, TD(\tau), m, 3)) \Rightarrow (\exists A: \text{sched}(A, TD2(\tau), m, 3)) \quad (3)$$

because the right hand side reduces the demand on processors.

We also have:

$$(\exists A: \text{sched}(A, TD(\tau), m, 3)) \Rightarrow (\exists A: \text{sched}(A, TD3(\tau), m, 3)) \quad (4)$$

because the right hand side reduces the demand on resources.

We can apply Theorem 2.2 from [5] on type-1 and typ-3 and we can also observe that on each processor of type-2 we perform non-preemptive EDF which is as good as is possible. Therefore, we have:

$$(\exists A: \text{sched}(A, TD1(\tau), m, 3)) \Rightarrow \text{sched}(\text{gEDF}, TD1(\tau), m, 6) \quad (5)$$

and

$$(\exists A: \text{sched}(A, TD2(\tau), m, 3)) \Rightarrow \text{sched}(\text{non-preemptive-EDF-on-each-processor-where-a-task-is-assigned-to-a-processor-based-on-the-resource-it-requests}, TD2(\tau), |R|, 3) \quad (6)$$

and

$$(\exists A: \text{sched}(A, TD3(\tau), m, 3)) \Rightarrow \text{sched}(\text{gEDF}, TD3(\tau), m, 6) \quad (7)$$

Combining (1)-(7) gives us:

$$(\exists A: \text{sched}(A, \tau, m, 1)) \Rightarrow \text{sched}(\text{gEDF}, TD1(\tau), m, 6) \quad (8)$$

and

$$(\exists A: \text{sched}(A, \tau, m, 1)) \Rightarrow \text{sched}(\text{non-preemptive-EDF-on-each-processor-where-a-task-is-assigned-to-a-processor-based-on-the-resource-it-requests}, TD2(\tau), |R|, 3) \quad (9)$$

and

$$(\exists A: \text{sched}(A, \tau, m, 1)) \Rightarrow \text{sched}(\text{gEDF}, TD3(\tau), m, 6) \quad (10)$$

Specifically, the equation (8) is obtained by combining (1), (2), (5). The equation (9) is obtained by combining (1), (3), (6). The equation (10) is obtained by combining (1), (4), (7).

Multiplying the processor speeds of (8),(9) and (10) by $m/(12m+3|R|)$ gives us:

$$(\exists A: \text{sched}(A, \tau, m, m/(12m+3|R|))) \Rightarrow \text{sched}(\text{gEDF}, TD1(\tau), m, 2m/(4m+|R|)) \quad (11)$$

and

$$(\exists A: \text{sched}(A, \tau, m, m/(12m+3|R|))) \Rightarrow \text{sched}(\text{non-preemptive-EDF-on-each-processor-where-a-task-is-assigned-to-a-processor-based-on-the-resource-it-requests}, TD2(\tau), |R|, m/(4m+|R|)) \quad (12)$$

and

$$(\exists A: \text{sched}(A, \tau, m, m/(12m+3|R|))) \Rightarrow \text{sched}(\text{gEDF}, TD3(\tau), m, 2m/(4m+|R|)) \quad (13)$$

Consider now the right-hand sides of (11),(12),(13). They state that deadlines are met when the new algorithm is used and they consider each processor type. Therefore, we have:

$$(\exists A: \text{sched}(A, \tau, m, m/(12m+3|R|))) \Rightarrow \text{sched}(\text{gEDF-vpr}, \tau, m, 1) \quad (14)$$

Multiplying the processor speed by $(12m+3|R|)/m$ and rewriting gives us:

$$(\exists A: \text{sched}(A, \tau, m, 1) \Rightarrow \text{sched}(\text{gEDF-vpr}, \tau, m, 12*(1+|R|/4m)) \quad (15)$$

This states the competitive ratio of the algorithm gEDF-vpr.

References

[1] L. Sha, R. Rajkumar and J. P. Lehoczky, Priority Inheritance Protocols: An Approach to Real-Time Synchronization. IEEE Transactions on Computers, September 1990, pp. 1175-1185.

[2] T. P. Baker, Stack-based Scheduling of Realtime Processes. Real-Time Systems 3(1): 67-99 (1991)

[3] C. L. Liu and J. W. Layland, Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment, Journal of the ACM (JACM), Volume 20 , Issue 1 (January 1973), Pages: 46 – 61, 1973.

[4] http://www.dagstuhl.de/Materials/Files/08/08071/08071.BaruahSanjoy.Abstract.txt

[5] C. A. Phillips, C. Stein, E. Torng, J. Wein, "Optimal time-critical scheduling via resource augmentation", Proceedings of the twenty-ninth annual ACM symposium on Theory of Computing, El Paso, Texas, United States, Pages: 140 – 149, 1997.